|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Data Structure** | **Time Complexity** | | | | | | | | **Space Complexity** |
|  | **Average** | | | | **Worst** | | | |  |
| **Access** | **Search** | **Insertion** | **Deletion** | **Access** | **Search** | **Insertion** | **Deletion** |
| **Array** | O(1) | O(n) | O(n) | O(n) | O(1) | O(n) | O(n) | O(n) | O(n) |
| **Stack** | O(n) | O(n) | O(1) | O(1) | O(n) | O(n) | O(1) | O(1) | O(n) |
| **Queue** | O(n) | O(n) | O(1) | O(1) | O(n) | O(n) | O(1) | O(1) | O(n) |
| **Single LL** | O(n) | O(n) | O(1)\* | O(1)\* | O(n) | O(n) | O(1)\* | O(1)\* | O(n) |
| **Doubly LL** | O(n) | O(n) | O(1)\* | O(1)\* | O(n) | O(n) | O(1) | O(1) | O(n) |
| **Hash Table** | NA | O(1) | O(1) | O(1) | NA | O(n) | O(n) | O(n) | O(n) |
| **BST** | O(logn) | O(logn) | O(logn) | O(logn) | O(n) | O(n) | O(n) | O(n) | O(n) |
| **Red-Black Tree** | O(logn) | O(logn) | O(logn) | O(logn) | O(logn) | O(logn) | O(logn) | O(logn) | O(n) |
| **AVL Tree** | O(logn) | O(logn) | O(logn) | O(logn) | O(logn) | O(logn) | O(logn) | O(logn) | O(n) |
| **B-Tree** | O(logn) | O(logn) | O(logn) | O(logn) | O(logn) | O(logn) | O(logn) | O(logn) | O(n) |
|  |  |  |  |  |  |  |  |  |  |

\*O(n), if operation is done at end of list.

**Sorting Algorithms**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Algorithm** | **Time Complexity** | | | **Space Complexity** |
|  | **Best** | **Average** | **Worst** |  |
| **Quick Sort** | O(nlogn) | O(nlogn) | O(n^2) | O(logn) |
| **Merge Sort** | O(nlogn) | O(nlogn) | O(nlogn) | O(n) |
| **Heap Sort** | O(nlogn) | O(nlogn) | O(nlogn) | O(1) |
| **Bubble Sort** | O(n) | O(n^2) | O(n^2) | O(1) |
| **Insertion Sort** | O(n) | O(n^2) | O(n^2) | O(1) |
| **Selection Sort** | O(n^2) | O(n^2) | O(n^2) | O(1) |
| **Bucket Sort** | O(n+k) | O(n+k) | O(n^2) | O(n) |
| **Radix Sort** | O(nk) | O(nk) | O(nk) | O(n+k) |
| **Counting Sort** | O(n+k) | O(n+k) | O(n+k) | O(k) |

**Searching Algorithms**

1. **Depth First Search:**
   1. **Time Complexity:** 
      1. O(V+E), Adjacency list
      2. O(V^2), Adjacency Matrix
   2. **Space Complexity:**
      1. O(V)
   3. **Applications**:
      1. Topological Sort
      2. Finding connected components
      3. Finding strongly connected components
2. **Breadth First Search:**
   1. **Time Complexity:**
      1. O(V+E), Adjacency list
      2. O(V^2), Adjacency Matrix
   2. Space Complexity:
      1. O(V)
   3. **Applications**:
      1. Finding all connected components in a graph
      2. Finding all nodes within one connected component
      3. Finding the shortest path between two nodes
      4. Testing of graph for bipartiteness
3. **Binary Search:**
   1. **Time Complexity:**
      1. Log(n), sorted array of n elements
   2. Space Complexity: O(1)
4. **Shortest Path by Dijkstra:**
   1. **Using Mean heap as priority queue:**
      1. **Time complexity:** O(ELogV), Where E is the number of updates for the heap.
      2. Space Complexity: O(V)
   2. **Using unsorted array as priority queue:** 
      1. **Time Complexity:** O(V^2)
      2. **Space Complexity:** O(V)
   3. Notes:
      1. Single Source shortest path
      2. It uses greedy method: Always pick the next closest vertex to the source.
      3. It uses priority queue to store unvisited vertices by distance from s.
      4. It does not work with negative weights
5. **Shortest path by Belmond ford:**
   1. **Time Complexity:** 
      1. O(E.V), if adjacency list is used.
   2. Space Complexity: O(V)
   3. Notes:
      1. Single source shortest path
      2. It works with negative weights as well.
      3. It uses Dequeue

**Binary Heap:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Heapify | FindMax | Extract Max | Increase Key | Insert | Delete | Merge |
| O(n) | O(1) | O(Logn) | O(logn) | O(logn) | O(logn) | O(m+n) |

**Minimal Spanning Tree:**

Spanning Tree is a graph having all vertices. We can have multiple spanning tree for a given graph. We have to find minimum spanning tree. That means adding all edges weight should be minimum.

1. **Prim’s Algorithm:**
   1. **Time Complexity:**
      1. O(V^2), without heap. Good for dense graph
      2. O(ElogV), with binary heap. Good for sparse graph
2. **Kruskal’s Algorithm:**
   1. **By using Disjoint sets (Unioin and Find):**
   2. **By using priority queue (Maintains weights in priority queue):**
      1. **Time Complexity:** O(ElogE)